

Algorithm: Lab7 (By Sujiv Shrestha ID:610145)

Problem 1.

1. [Interview Question] Devise an $O(n)$ algorithm to accomplish this task:

Given a none-empty string S of length n , S consists some words separated by spaces. We want to reverse every word in S .

For example, given $S = \text{"we test coders"}$, your algorithm is going to return a string with every word in S reversed and separated by spaces. So the result for the above example would be "ew tset sredoc" .

Algorithm reverseWord(S)

Input string S with n characters

Output string $sRet$ with n characters with each word reversed

$sRet \leftarrow ""$

$stkList \leftarrow \text{create List of Stack of characters}$

$cnt \leftarrow 0$

for $i \leftarrow 0$ to $n-1$ **do**

if ($S[i] = ' '$) **then**

$cnt \leftarrow cnt + 1$

else

if ($stkList.size \leq cnt$) **then**

$stkList.add(\text{emptyStack})$

$stkList.get(cnt).push(S[i])$

while ($!stkList.isEmpty$) **do**

$stack \leftarrow stkList.removeFirst$

while ($stack.isEmpty$) **do**

$sRet \leftarrow sRet \cup stack.pop()$

$sRet \leftarrow sRet \cup ' '$

return $sRet$

Running Time, $T(n) = O(n) + O(n/k * k)$
 $= O(n) + O(n)$
 $= O(n)$

Java Implementation using Stack

```
public static String reverseWord(String s) {
    String sRet = "";
    List<Stack<Character>> stkList = new ArrayList<>();
    int i=0;
    stkList.add(new Stack<Character>());
    for(Character c:s.toCharArray()) {
        if(c==' ') {
            i++;
            continue;
        }
        if(stkList.size()<i+1)
            stkList.add(new Stack<Character>());

        stkList.get(i).push(c);
    }
}
```

```

    }
    for(Stack<Character> stk:stkList) {
        while(!stk.isEmpty()) {
            sRet = sRet+stk.pop();
            StringBuilder sb = new StringBuilder();
            sb.append(stk.toArray());
        }

        sRet = sRet+" ";
    }
    return sRet.substring(0,sRet.length()-1);
}

```

Problem2

2. Create a sorting routine based on a BST and place it in the sorting environment, distributed earlier. For this, your new class, BSTSort, should be a subclass of Sorter. Your BSTSort class can be essentially the same as the BST class given in the slides (see the folder in your labs directory for this lab), except that you will need to modify the printTree method so that it outputs values to an array (rather than printing to console).

Solution:

//printTr method in MyBST.java

```

    public int[] printTree() {
        if(root==null)
            return new int[0];
        else {
            List<Integer> sortedLst = printTr(root);
            int[] sorted = new int[sortedLst.size()];
            for(int i=0;i<sortedLst.size();i++) {
                sorted[i] = sortedLst.get(i);
            }
            return sorted;
        }
    }

    private List<Integer> printTree(Node t) {
        List<Integer> retIntList = new ArrayList<Integer>();
        if (t != null) {
            retIntList.addAll(printTr(t.left));
            retIntList.add(t.element);
            retIntList.addAll(printTr(t.right));
        }
        return retIntList;
    }
}

```

//My BSTSort.java class

```

package sortroutines;

import java.util.Arrays;
import runtime.Sorter;

public class BSTSort extends Sorter {

```

```

    public static void main(String[] args) {
        BSTSort bst = new BSTSort();
        System.out.println(Arrays.toString(bst.sort(new int[] {0,2,1,5,3,7,9,8})));
    }

    @Override
    public int[] sort(int[] arr) {
        MyBST myBST = new MyBST();
        for(int x:arr) {
            myBST.insert(x);
        }
        return myBST.printTr();
    }
}

```

Output:

[0, 1, 2, 3, 5, 7, 8, 9]

After you have implemented, discuss the asymptotic running time of your new sorting algorithm. Run an empirical test in the sorting environment and explain where BSTSort fits in with the other sorting routines (which algorithms is it faster than? which is it slower than?).

Answer:

When running for all the sorting algorithms for input array of size range 1000-15000 following result was observed:

```

40 ms -> MergeSortPlus
49 ms -> MergeSort
64 ms -> QuickSort
160 ms -> BSTSort
479 ms -> InsertionSort
845 ms -> SelectionSort
5713 ms -> BubbleSort

```

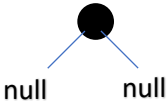
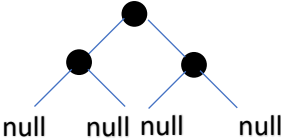
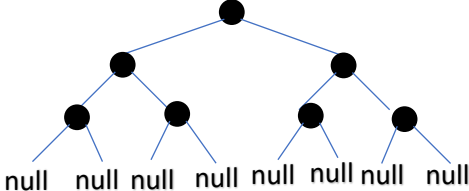
It is observed that BSTSort is better than BubbleSort, InsertionSort and SelectionSort. While most of the time it is closer to QuickSort. So, it is slower than MergeSort and QuickSort.

Problem3

3. For each integer $n = 1, 2, 3, \dots, 7$, determine whether there exists a red-black tree having exactly n nodes, with *all of them black*. Fill out the chart below to tabulate the results:

Num nodes n	Does there exist a red-black tree with n nodes, all of which are black?
1	Yes
2	No
3	Yes
4	No
5	No
6	No
7	Yes

Red-Black Tree examples:

<p>$n=1$</p> 	<p>$n=3$</p> 
<p>$n=7$</p> 	

Problem4

4. For each integer $n = 1, 2, 3, \dots, 7$, determine whether there exists a red-black tree having exactly n nodes, where *exactly one of the nodes is red*. Fill out the chart below to tabulate the results:

Num nodes n	Does there exist a red-black tree with n nodes, exactly one of the nodes is red?
1	No
2	Yes
3	No
4	Yes
5	No
6	No
7	No

